Carnegie Mellon
Software Engineering Institute

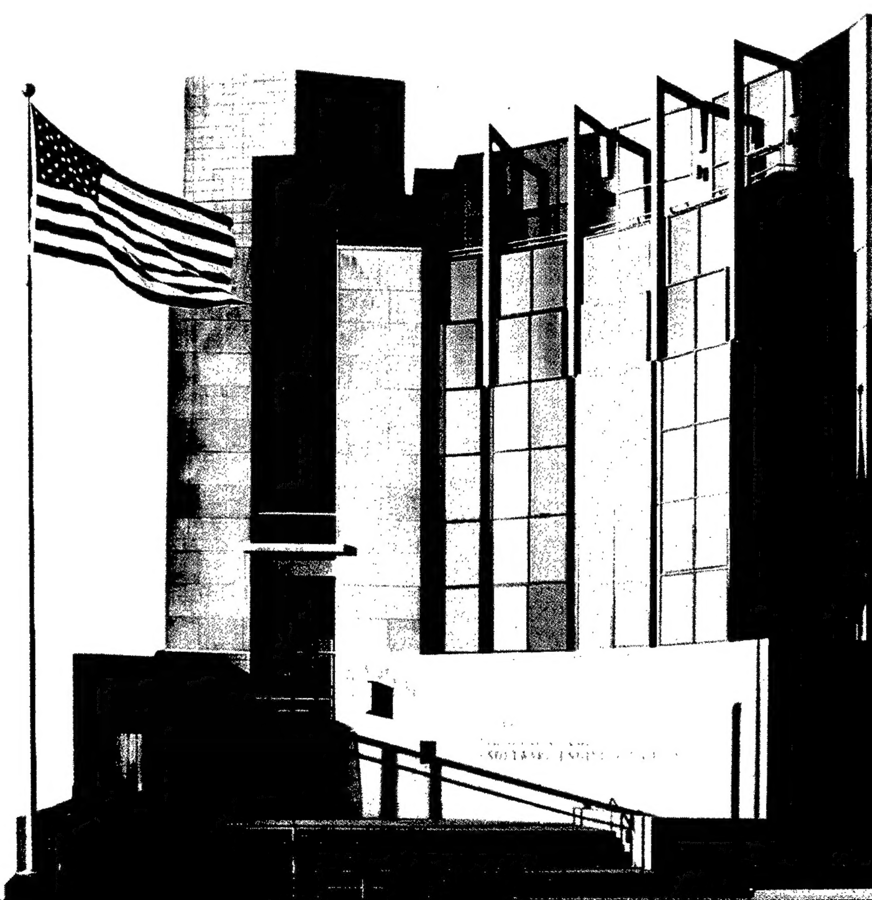
A Federation Object Model (FOM) Flexible Federate Framework

Regis Dumond
Reed Little

March 2003

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2003-TN-007



A Federation Object Model (FOM) Flexible Federate Framework

Regis Dumond
Reed Little

March 2003

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2003-TN-007

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

Abstract

The concept of a framework as a reusable software component has become a state-of-the-practice technique in software companies. A number of frameworks based on High Level Architecture (HLA) are available commercially, and many companies have developed their own frameworks for internal applications. Using a framework reduces development time and allows software architects and programmers to focus on the unique aspects of the simulation. However, the challenge of developing a reusable component to support dynamic reconfigurability remains. Indeed, existing frameworks use a static object-model representation that requires full knowledge of object model components when a federate is built (at compile-time). This report describes an approach to designing a domain framework that encapsulates expertise in developing an HLA federate by hiding runtime infrastructure (RTI) internal operations from the developer. This approach uses a Java™ virtual machine and a parser to map object representations of federation object model (FOM) elements dynamically.

1 Introduction

1.1 Background and Approach

The object model defined by High Level Architecture (HLA) Object Model Template (OMT) specifications is a foundation for promoting interoperability between simulations [IEEE 00a]. The chief advantage of implementing an HLA framework lies in software reusability and productivity. Implementing an HLA framework implies that the framework design must be independent of any particular object model [Cox 98].

A federate software framework may provide two fundamental methods for object model representation:

1. static object-model representation, which requires full knowledge of object components when a federate is built (at compile-time)
2. dynamic object-model representation, which is determined at runtime

This report describes a new object-oriented framework currently in development that will support the reconfigurability of a simulation object model (SOM) or federation object model (FOM) at runtime. Our approach is based on dynamic representation using the concept of meta-representation. This concept is more flexible than static representation in reconfiguring a federate dynamically. We also use a set of design patterns to encapsulate and hide runtime infrastructure (RTI) mechanisms. Doing so reduces the development time required to adapt simulation applications to the HLA.

1.2 Overview of the High Level Architecture

The HLA was developed by the U.S. Department of Defense (DoD) in response to the value of simulation to support a wide variety of military applications, as well as the need to manage simulations to ensure that they provide a cost-effective tool. In particular, simulations developed for one purpose can be readily reused in other applications, either individually or in combination. This reuse and interoperability is critical if simulations are to be affordable and useful to the changing needs of the DoD.

A preliminary version of the HLA was published in 1995, with the final DoD version published in 1996. Two Institute of Electrical and Electronics Engineers (IEEE) versions of the HLA standard were published in 2000 [IEEE 00a, IEEE 00b].

The HLA is a technical architecture developed to facilitate the reuse and interoperation of simulations. It is based on the premise that no simulation can satisfy all uses and users. An individual simulation or set of simulations developed for one purpose can be applied to another application under the HLA concept of the federation—a composable set of interacting simulations. The intent of the HLA is to provide a structure that will support the reuse of capabilities available in different simulations, ultimately reducing the cost and time required to create a synthetic environment for a new purpose and providing developers with the option of distributed collaborative development of complex simulation applications.

The HLA has wide applicability, across a full range of simulation application areas, including education and training, analysis, engineering, and even entertainment, at a variety of resolution levels. These widely differing application areas indicate the variety of requirements that have been considered in the development and evolution of the HLA.

The HLA does not prescribe a specific implementation or mandate the use of particular software or programming languages. Over time, as technology advances become available, new implementations will be possible within the HlA's framework.

1.3 Functional Synopsis of the HLA

Figure 1 shows the major functional components of an HLA federation. The ovals represent a set of simulations, or more generally, the federates. Federates can be event-driven simulations, real-time human-in-the-loop simulators, live equipment, supporting utilities (such as stealth viewers or data collectors), or even interfaces to live players or instrumented facilities. The HLA imposes no constraints on what is represented in the federates or how it is represented, but it does require that all federates incorporate specified capabilities to allow federates to interact with other federates. This incorporation is achieved through the exchange of data supported by services implemented in a common federate communications interface (referred to as an RTI). The RTI is, in effect, a common communications interface for the federation. It provides a general-purpose set of services that support the federates in carrying out the federate-to-federate interactions and federation management support functions. These services will be discussed in Section 1.3.1. All interactions among the federates go through the RTI.

The federates use the RTI via a runtime interface. The HLA runtime interface specification provides a standard way for federates to interact with the RTI, to invoke the RTI services to support runtime interactions among federates, and to respond to requests from the RTI. This interface is independent of the implementation, specific object models, and data-exchange requirements of any federation.

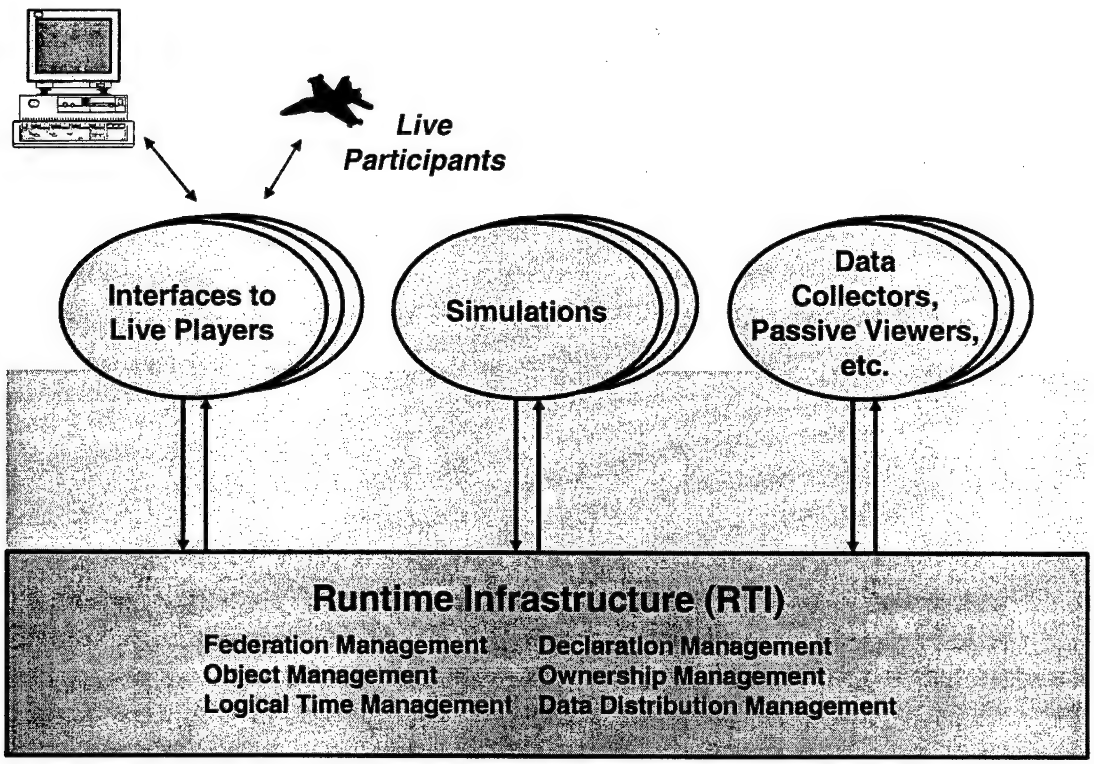


Figure 1: Functional View of an HLA Federation

The HLA standard is formally defined by three documents:

1. the interface specification
2. the object model template
3. the HLA rules

1.3.1 HLA Interface Specification

The HLA interface specification describes the runtime services provided to the federates by the RTI and to the RTI by the federates. There are seven classes of services:

1. *Federation management* services offer basic functions required to create and operate a federation.
2. *Declaration management* services support the efficient management of data exchange through the information provided by federates. Such information defines the data that the federates will provide and require during a federation execution.
3. *Object management* services provide creation, deletion, identification, and other services for the actual transfer of data.
4. *Ownership management* services support the dynamic transfer of ownership of HLA object-instance attributes during an execution.
5. *Time management* services support the synchronization of runtime federate data exchange.

6. *Data distribution management* services support the efficient routing of data among federates during the course of a federation execution.

7. Other miscellaneous services are also provided.

The HLA interface specification defines how these services are accessed, both functionally and in an application program interface (API). APIs that are currently available include those for Common Object Request Broker Architecture (CORBA), Interface Description Language (IDL), C++, Ada-95, and Java™.

1.3.2 HLA Object Models

HLA object models are descriptions of the essential sharable elements of a federate or federation in "object-based" terms. The HLA is directed towards interoperability; hence in the HLA, object models are intended to focus on describing the critical aspects of federates and federations that are shared across a federation. The HLA puts no constraints on the content of the object models. However, the HLA does require that each federate and federation document its object model using a standard object-model template. These templates are intended to be the means for open information-sharing across the community to facilitate the reuse of federates, and the completed templates are often openly available. In addition, some tools exist (and more are being developed) to allow for automated searches and reasoning about object-model template data and to further facilitate cost-effective information exchange and reuse.

The HLA specifies two types of object models: the HLA FOM and the HLA SOM. The HLA FOM describes the set of object classes, attributes, and interaction classes that are shared across a federation. The HLA SOM describes the federate in terms of the categories of object classes, attributes, and interaction classes that it can offer to federations. The SOM is distinct from internal federate design information. It provides information on the capabilities of a federate to exchange information as part of a federation. The SOM is essentially a contract by the federate defining the sorts of information it can make available in future federations. The SOM facilitates the assessment of whether it is appropriate for the federate to participate in a federation.

While the HLA does not define the contents of an SOM or FOM, it does require that a common documentation approach be used. Both the HLA FOM and SOM are documented using a standard format and syntax called the HLA OMT.

1.3.3 HLA Rules

Finally, the HLA rules summarize the key principles behind the HLA. These rules are divided into two groups: federation and federate rules. Federations, or sets of interacting federates,

™ Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

are required to have an FOM in the OMT format. During runtime, all data representation takes place in the federates (not in the RTI) with only one federate owning any given attribute of an HLA object instance at any given time. Information exchange among the federates takes place via the RTI as specified in the HLA interface specification.

Additional rules apply to individual federates. Under the HLA, all federates must document the public information they provide to and require from the federation in their SOM using the OMT. As documented in their SOM, federates must import and export information, transfer attribute ownership, and interact with the time management services of the RTI when managing local time.

2 Object-Oriented Framework

2.1 Definition

An object-oriented framework is a reusable software architecture made of both design and code. It represents a partial design and implementation for an application in a given problem domain. Johnson and Foote have developed the most frequently used definition:

> *A framework is a set of classes that embodies an abstract design for solutions to a family of related problems* [Johnson 88].

An object-oriented framework is characterized by a set of classes and control flows and provides a precise structure for defining new applications.

2.2 White-Box Versus Black-Box Framework

Frameworks that can be extended only by inheritance are called white-box frameworks, while frameworks that can be extended by enhancing existing components are called black-box frameworks. Black-box frameworks are easier to use since the internal mechanism is hidden from the developer. Our framework is a black-box type that enables users to customize behavior by configuring components.

2.3 Inversion of Control

In the traditional class library, the application code invokes routines in the library. By comparison, a framework has the thread of control and calls the application code when appropriate. In our case, the framework calls objects in the federate when they are needed.

2.4 Metamodel and Framework

A metamodel describes the structure of an application, a process, or a group of related objects. An object metamodel might define object types, their attributes, the types of those attributes, and the relationships allowed between the objects.

These descriptions are stored in concise formats that can be processed by a framework. The framework interprets the information and makes decisions without adding or modifying code. Using a metamodel, our framework can load data from an SOM/FOM file and reconstruct objects in memory, without recompiling the code.

3 Metarepresentation

One way to develop a framework that supports multiple FOMs without code modification is through the concept of metaclass, as shown in Figure 2. If a class describes objects, we can distribute classes and instances onto two hierarchical levels. Instances belong to the lower level, while models or classes belong to the upper level. To design the class notion, it is necessary to have a supplementary level containing the models that describe the classes. The notion of metaclass consists of applying this modeling to the classes themselves. A metaclass is realized by describing, in a particular class (meta), the attributes and methods that characterize a class.

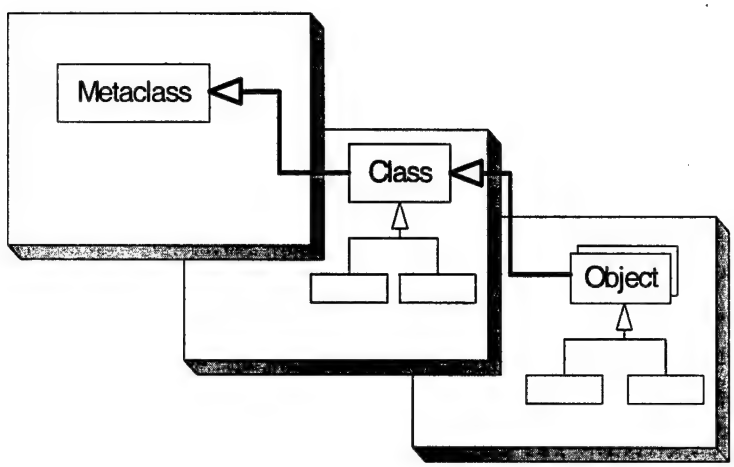


Figure 2: Representation of Three System Levels

Although the HLA is not strictly object oriented, it provides a two-level system that defines object classes and instances with distinct identities. We introduced the third level using the descriptions of object and interaction class structure defined in the object model template (OMT). With this definition, we created a metaclass representation of elements that compose the FOM/SOM. The Unified Modeling Language (UML) class diagram in Figure 3 shows this representation [Booch 99].

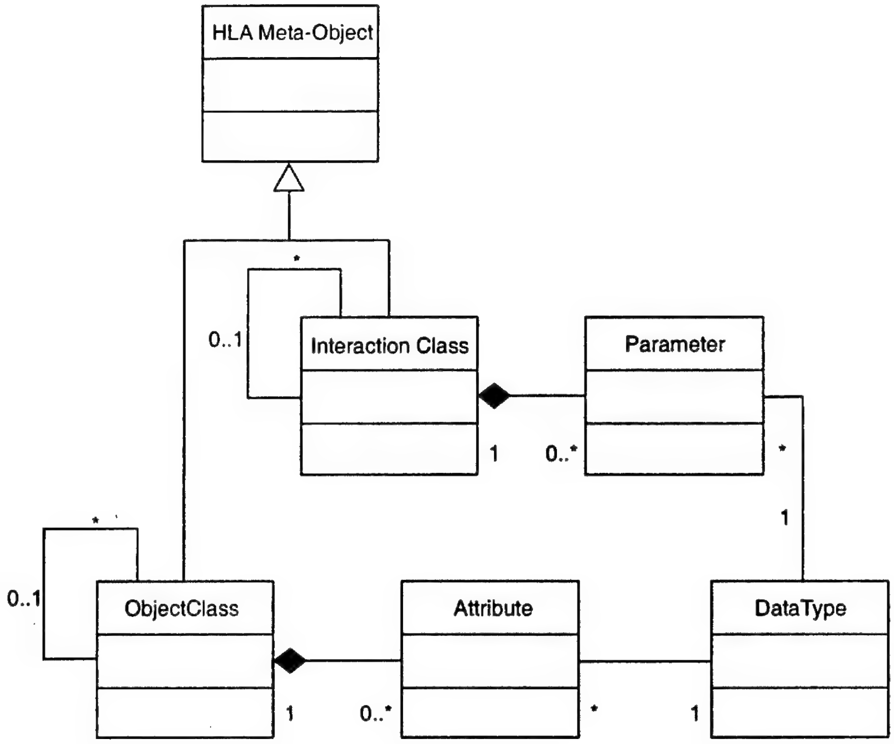


Figure 3:   Metaclass Representation

Metaclasses are entities located at a meta-level with regard to instance and classes [Masini 89]. Classes are instances of a metaclass, and classes have properties that may be complex, and therefore, difficult to generalize. However, in the HLA, the number of possible types for such properties is finite. The OMT defines classes of objects, each having a name and a set of data called attributes. Classes of defined objects form a single inheritance tree, and its properties are used to build the metamodel. An object class model provides a set of useful information about the federate, like subscribable or publishable that can be inserted into a metaclass.

Attribute and parameter models are also converted to appropriate meta-attributes. Attribute metadata are composed of a name, a set of properties (e.g., ownership, transport and delivery order, and update/reflect permission), and a data type. A data type is metadata that represents the notion of value type common to both OMT attributes and parameters.

Figure 4 shows how the data types are represented within the framework.

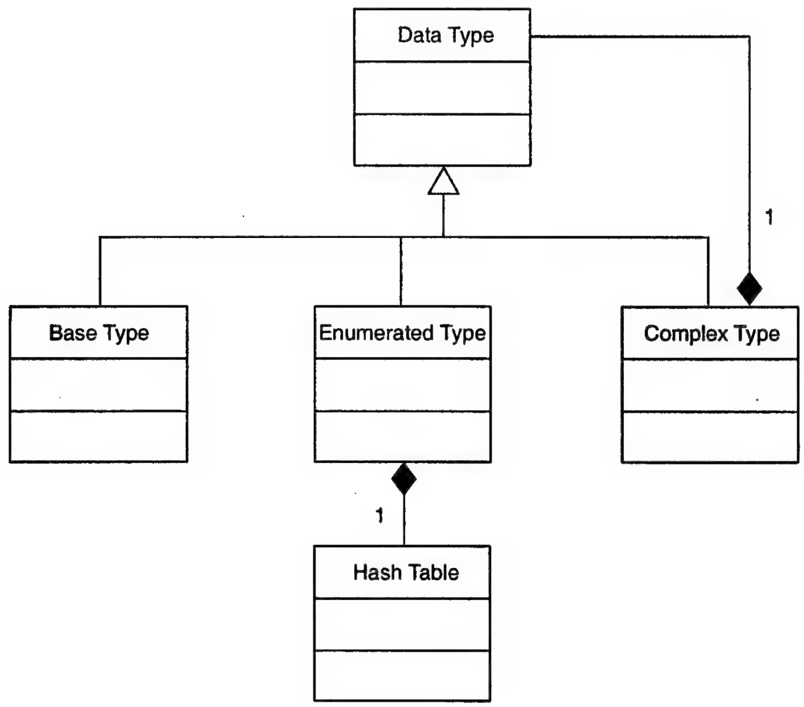


Figure 4: Metadata Representation

We use the following Java mechanisms and design patterns to fix value types at runtime:

- The base types are built with classes provided in Java to represent the primitive type (e.g., Integer, Byte, Double, Float, Long, and Short) and String. In this case, Java is particularly useful for creating a base type at runtime and fixing its value.
- Enumerated types use a hash table structure to associate a name to a value.
- Complex types are represented by the pattern composite, which composes objects (Data Types) into tree structures to represent part-whole hierarchies [Gamma 95].

This metamodel is the foundation of our framework. We use it to translate the FOM/SOM details into a repository for the framework object model at runtime and make its content available through intuitive interfaces.

A parser was written to process an object model to map the FOM/SOM information into node structures used to build the repository for the object model. The HLA OMT specification [IEEE 00a] defines the FOM/SOM with the Extensible Markup Language (XML).

4 Overview of the Framework Architecture

The framework is a set of Java classes that provide a layer between the federate application and a portable Java RTI (pRTI™) [Pitch 02].

The Exploration Edition of pRTI is a multi-threaded RTI. It does not require the federate to call a `tick()` function periodically.

Figure 5 presents an overview of the framework.

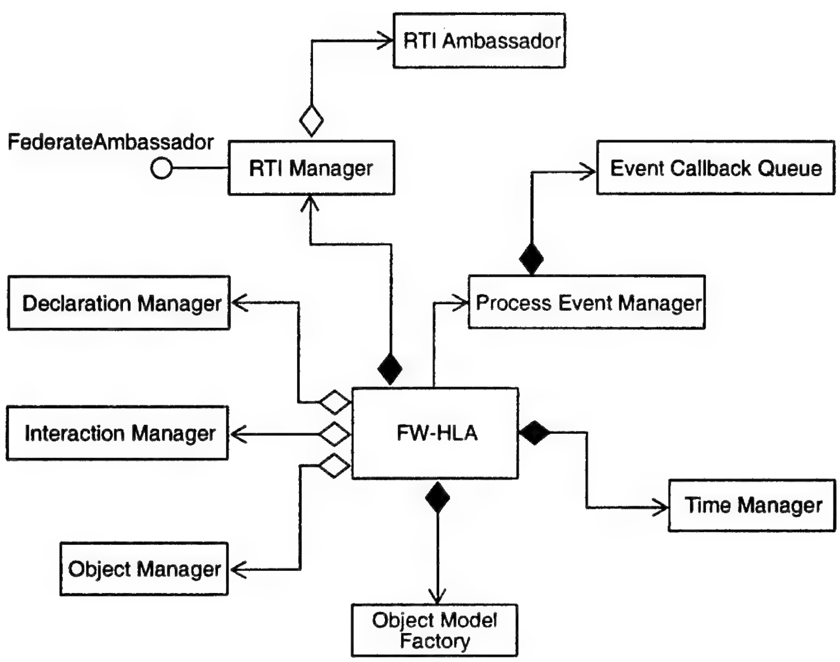


Figure 5: Overview of the Framework Architecture

The framework is composed of different modules:

- FW-HLA manages module activities and provides an interface to configure and use the framework with a federate.
- Object Model Factory uses an SOM/FOM file to build a meta-representation of an HLA object model.
- RTI Manager is the communication layer between the RTI and the framework.
- Declaration Manager publishes and/or subscribes object attributes and interactions.
- Object Manager manages local and remote proxies, registers objects, and updates attributes values.

- Interaction Manager sends interactions.
- Time Manager coordinates event activities within the framework.
- Process Event Manager manages all events, scheduled or not, within the framework.

The following sections describe the mechanisms used inside each module.

4.1 FW-HLA Module

The FW-HLA module represents the framework entry point for federate developers. It loads a configuration file containing runtime parameters (e.g., SOM/FOM file location, federation name, federate name, and Time Management parameters) to initialize the federate.

It also provides a set of methods to

- create, join, and resign a federation
- publish or subscribe object attributes and interaction (i.e., Declaration Manager)
- register listeners on RTI callbacks (e.g., ReceiveInteraction, RemoveObject) to be notified each time an event occurs
- send interactions, create or delete object proxy instances, and update attributes (scheduled or not)
- query remote or local object proxy
- modify Time Management parameters (e.g., time constrained/regulated, look-ahead, and time step)

4.2 Object Model Factory Module

The Object Model Factory module uses the parser to read an FOM/SOM file. A metaclass is created for each object and interaction class definition. These metaclasses are maintained in a tree to provide quick lookup. The root of this object tree is the ObjectRoot node from the OMT file. Similarly, the InteractionRoot node is the root to the interaction tree. The Object Model Factory can create a specific kind of metaclass based on either a string name or an integer identifier for the class type as shown in Figure 6. When the Object Manager or Interaction Manager wants to create a proxy object, the Object Model Factory finds the metaclass under the specified name in the corresponding tree, copies it, and passes the copy back to the requester.

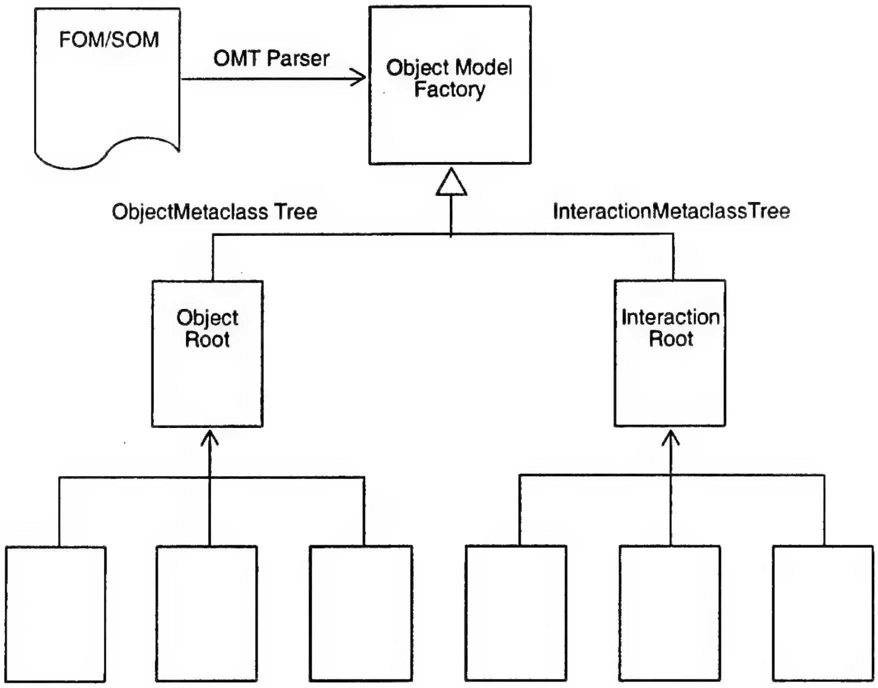


Figure 6: Object Model Factory

4.3 RTI Manager

The RTI Manager is a layer above the RTI. It implements the FederateAmbassador interface and provides an event mechanism to notify many objects or components of invocations about RTI-initiated services.

The Event Generator Java idiom enables interested objects (listeners) to be notified about states or events experienced by an "event generator" [Venners 02].

The RTI Manager implements this idiom to invert dependencies between the FederateAmbassador and the components included in the framework. Figure 7 depicts the structure of an example based on the reception of the DiscoverObject RTI callback.

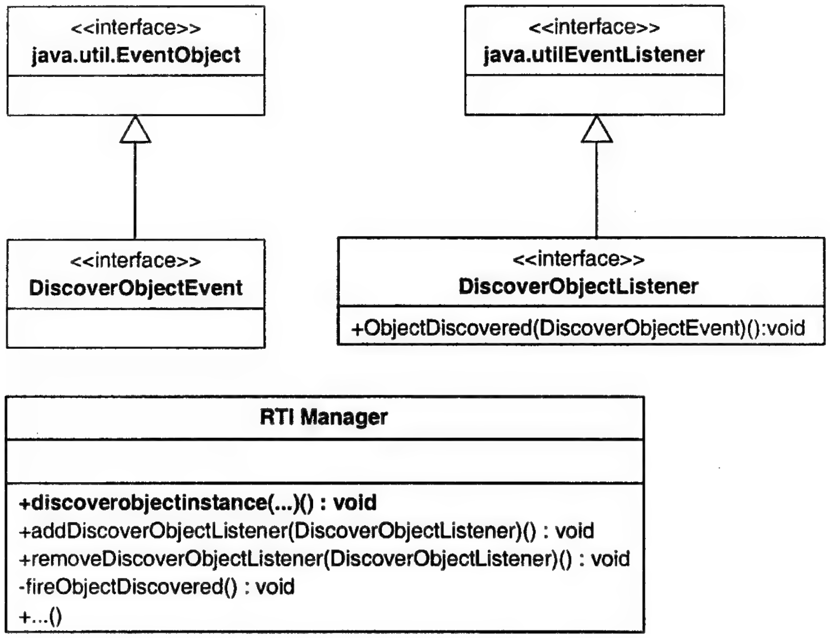


Figure 7: Event Generator Example

When the DiscoverObject callback occurs, the RTI Manager creates a DiscoverObjectEvent containing the information that is needed and propagates that information to each registered listener, as shown in Figure 8.

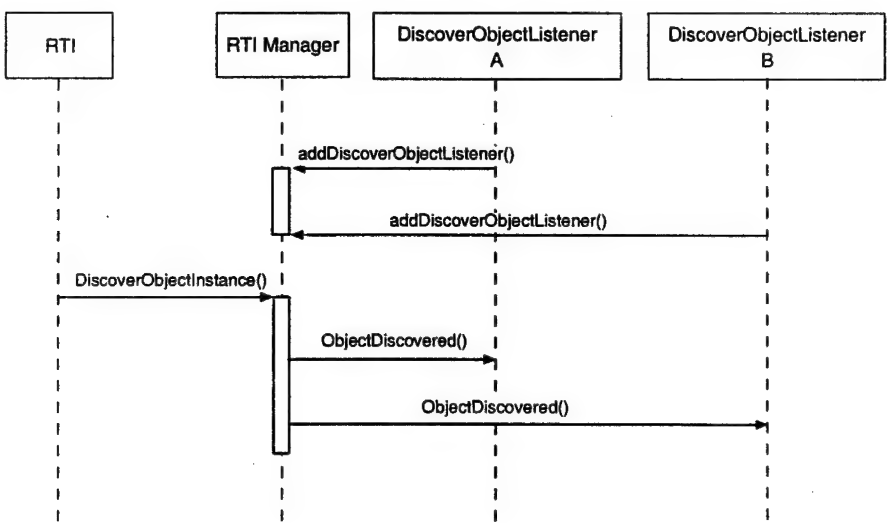


Figure 8: UML Sequence Diagram of Event Generator Example

For each category of callbacks in the FederateAmbassador, the framework provides an Event/Listener pair. This functionality introduces a variability point for adding future features or extensions. In this way, a component may be added to the framework by subscribing and implementing the interface listener, which corresponds to a special event provided by the RTI Manager.

The RTI Manager also encapsulates the RTI ambassador services to ensure compatibility with types defined in the framework. Additionally, the RTI Manager performs new services to resolve thread concurrency problems.

The RTI implementation used in the framework is multi-threaded; that is, the RTI-initiated services occur in a thread of control that is separate from those held by the framework. Dr. Kuhl recommends using the Barrier—an advanced thread pattern presented by Doug Lea in Concurrent Programming in Java [Lea 96]—to coordinate activities of the main thread with other threads [Kuhl 00]. The Barrier pattern is designed to put the main thread to sleep until a signal is given. We use the Barrier wherever the main thread invokes an RTI service and must wait for a corresponding RTI callback. For example, when the main thread enables the time constraint by invoking the EnableTimeConstrained callback, the main thread doesn't do anything until the RTI invokes the TimeConstrainedEnabled callback. In this case, instead of waiting in a loop for a variable state changeover and wasting computation, the RTI Manager uses the Barrier to put the main thread to sleep until the callback thread notifies the Barrier's state to wake it.

4.4 Declaration Manager

The Declaration Manager is responsible for publishing and/or subscribing attribute classes and interactions. Declarations to the RTI are made automatically using information contained in the metaclass trees. The users can subscribe or publish interaction or object classes through three different modes:

1. publishing or subscribing to everything (The Declaration Manager subscribes or publishes all classes defined in the SOM or FOM. This mode is particularly useful when the framework is used to develop a logger.)
2. examining the publish/subscribe capabilities of the attribute or interaction contained in the metaclass trees
3. publishing or subscribing a class by specifying its String name

4.5 Object Manager

The Object Manager maintains a list of local and remote proxies. An Object Proxy is a mechanism commonly used in distributed object technology. It represents a federation's object and mirrors its public attributes. When an HLA object instance is discovered in a federation or registered by the framework, the Object Manager creates a proxy by cloning the metaclass associated with the instance type and inserting it into a list.

Once the proxy is created, the Object Manager updates the attributes of local objects or reflects the attributes of remote objects automatically. The framework allows a developer to update an object attribute by invoking a method with the name and new value as arguments.

The framework fixes the value into the local object proxy and then sends the attribute updates to the RTI when the developer invokes the `Update()` method.

In the same way, when the RTI reflects attribute values, the framework retrieves the object proxy associated in the list and automatically stores the new values. The developer can then query remote object proxies locally and obtain their current attribute values.

4.6 Interaction Manager

The framework uses the Interaction Manager to send interactions to the RTI. As with the Object Manager, the process required to prepare the request to the RTI is performed automatically and hidden from the user. To send an interaction to the federation, the federate developer must use an instance of a specific interaction metaclass, fix the parameter values, and pass the instance to the framework.

The Interaction Manager implements an event mechanism (i.e., RTI Manager) to notify all registered listeners when an interaction occurs. It also maintains a queue of all interactions received, which may be consulted by the federate.

4.7 Time Manager

The Time Manager uses the RTI management services to synchronize time advancement of the framework among federates in a federation. A federate asks to advance its time and waits for a callback from the RTI to grant it.

The Time Manager currently supports two types of time management simulation:

1. Time-stepped simulation uses the Time Advance Request service to advance time.
2. Event-driven simulation requests an advance of its clock by invoking the Next Event Request service.

The Time Manager uses an advanced thread mechanism (i.e., RTI Manager) to await the reception of a TimeAdvanceGrant callback from the RTI.

For example, when the framework uses an event-driven simulation, it seeks to advance its time to the time of the smallest event contained in the priority queue of the Process Event Manager. When the time advance request is granted, the RTI Manager determines if the logical time is greater or smaller than the requested time. If it is greater, the RTI Manager signals the Process Manager to process the corresponding callback events. If the logical time is smaller than the Next Event Request time, the Time Manager waits until the requested time expires to avoid a callback event with an earlier logical time.

In the same way, a time-stepped simulation uses the Time Advance Request to move its logical time forward in equal steps. The size of the step is a parameter of the configuration file, but it may be modified at runtime by a method provided by the FW-HLA module.

4.8 Process Event Manager

The Process Event Manager is responsible for invoking all actions or modification inside the framework. In general, each RTI service is implemented by a corresponding event callback, as shown in Figure 9.

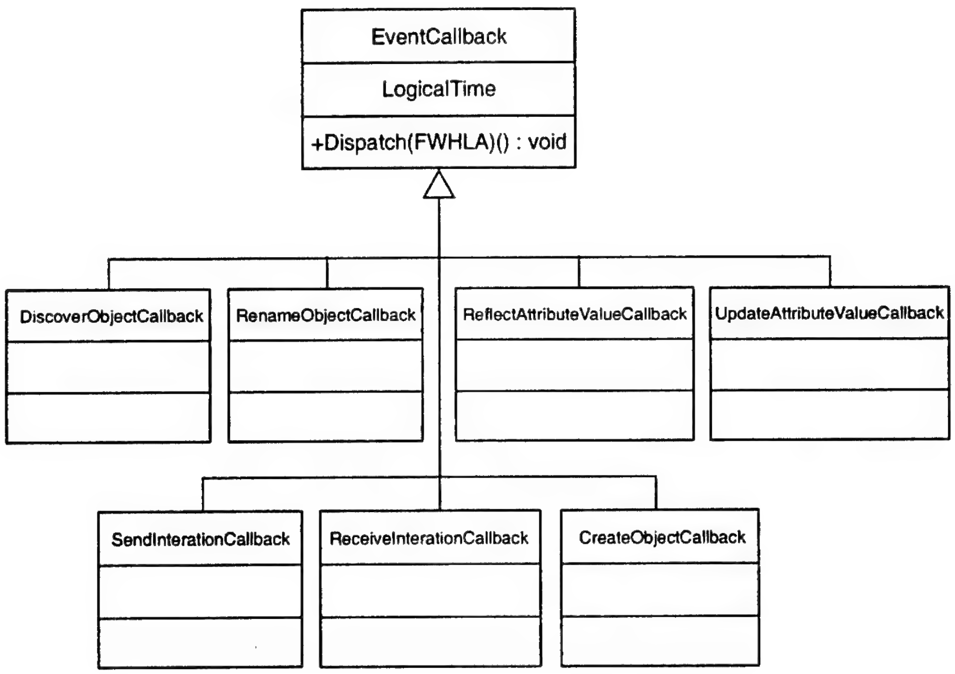


Figure 9: Process Event Callbacks

An event callback is composed of a LogicalTime attribute, representing an execution time, and a procedure containing a set of associated behaviors.

An execution time specifies when the Process Event Manager must invoke the procedure code. For example, an UpdateAttributes event can be scheduled to execute at a time specified by the federate.

The Process Event Manager implements a set of listener interfaces associated to RTI-initiated services (see Section 4.3, RTI Manager). In response to callback RTI events or a message from the federate (CreateObject, UpdateAttributes, etc.), the Process Event Manager creates an event callback based on the message and places it in a priority queue of scheduled event callbacks, ordered by increasing time. When the Process Event Manager receives a signal

(supplying a time *t*) from the Time Manager, it processes the event callback in correct time order. The sequence diagram in Figure 10 shows an example.

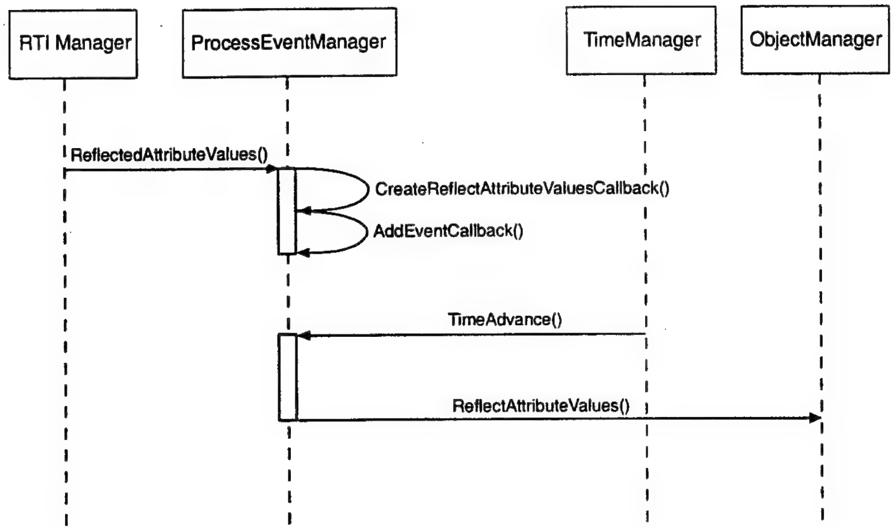


Figure 10: Process Event Example

An event callback performs a set of actions that calls functions within components from the framework. A ReflectAttributesEventCallback inserts methods into the Object Manager for updating a remote object proxy, but the developers are free to create or extend an event callback by providing a new behavior.

Like the RTI Manager, the Process Event Manager may be accessed from several threads. It uses a monitor thread mechanism to coordinate its queue activities between callback instances placed on the queue in response to the callback thread from RTI and actions that were removed from the queue in response to the Time Manager Thread [Lea 96].

5 Future Enhancements

The HLA framework is currently a work in progress. Several enhancements are planned:

- Make the architecture compliant with IEEE Standard 1516.1, *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification* [IEEE 00b].
- Incorporate retracting event features in the Time Manager.
- Support the Data Distribution Management (DDM).
- Add Management Object Model functionalities to obtain information about the federation and each federate.

6 Conclusion

The framework provides a flexible architecture that enables software designers to participate in a federation using different SOMs or FOMs without important code changes. However, developers should strike a balance between flexibility and performance before using this architecture. The framework's dynamic representation may decrease performance when a federate manages many objects. Message traffic on the event mechanism may generate a bottleneck. Developers should evaluate the federate's requirements to verify that the framework architecture can be inserted into the federate application.

References

[Booch 99] Booch, G.; Rumbaugh, J.; & Jacobson, I. *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley, 1999.

[Cox 98] Cox, Kevin. "A Framework-Based Approach to HLA Federate Development." *Proceedings of the 1998 Fall Simulation Interoperability Workshop*, Sept. 14-18, 1998. Orlando, FL: Simulation Interoperability Standards Organization (SISO), 1998. <http://www.sisostds.org/siw/98fall/ViewPaper_98F.htm>

[Gamma 95] Gamma, E. et al: *Design Patterns, Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[IEEE 00a] Institute of Electrical and Electronics Engineers. *IEEE Standard 1516.2-2000, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification*. New York, NY: Institute of Electrical and Electronics Engineers, 2000.

[IEEE 00b] Institute of Electrical and Electronics Engineers. *IEEE Standard 1516.1-2000, Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*, New York, NY: Institute of Electrical and Electronics Engineers, 2000.

[Johnson 88] Johnson, R. & Foot, B. "Designing Reusable Classes." *Journal of Object-Oriented Programming 1*, 2 (June/July 1988): 22-35.

[Kuhl 00] Kuhl, F.; Weatherly, R.; & Dahmann, J. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Upper Saddle River, NJ: Prentice Hall, 2000 (ISBN 0-13-022511-8).

[Lea 96] Lea, Douglas. *Concurrent Programming in Java: Design Principles and Patterns*. Reading, MA: Addison-Wesley, 1996.

[Masini 89] Masini G.; Napoli, A.; Colnet, D.; Leonard, D.; & Tombre, K. *Les Langages a Objets*. Paris, France: InterEdition, 1989 (ISBN 2-7296-0275-5).

[Pitch 02] Pitch. *Portable Runtime Infrastructure (pRTI)* <http://www.pitch.se/prti/> (valid as of December 2002).

[Venners 02] Venners, B. *Event Generator Idiom* <http://www.artima.com /designtechniques/eventgen.html> (valid as of December 2002).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE March 2003	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE A Federation Object Model (FOM) Flexible Federate Framework	5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Regis Dumond, Reed Little	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2003-TN-007
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS	12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The concept of a framework as a reusable software component has become a state-of-the-practice technique in software companies. A number of frameworks based on High Level Architecture (HLA) are available commercially, and many companies have developed their own frameworks for internal applications. Using a framework reduces development time and allows software architects and programmers to focus on the unique aspects of the simulation. However, the challenge of developing a reusable component to support dynamic reconfigurability remains. Indeed, existing frameworks use a static object-model representation that requires full knowledge of object model components when a federate is built (at compile-time). This report describes an approach to designing a domain framework that encapsulates expertise in developing an HLA federate by hiding runtime infrastructure (RTI) internal operations from the developer. This approach uses a Java™ virtual machine and a parser to map object representations of federation object model (FOM) elements dynamically.

14. SUBJECT TERMS High Level Architecture (HLA), reusable software component, framework, object model, federate, federation object model (FOM)	15. NUMBER OF PAGES 30
16. PRICE CODE	

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102